DIVE – a Multi-User Virtual Reality System

Christer Carlsson Olof Hagsand
cc@sics.se olof@sics.se
Swedish Institute of Computer Science
Box 1263
164 28 Kista, Sweden
tel: +46 8 752 1500
fax: +46 8 751 7230

Abstract

The Distributed Interactive Virtual Environment (DIVE) is a heterogeneous distributed VR system based on UNIX and Internet networking protocols. Each participating process has a copy of a replicated database and changes are propagated to the other processes with reliable multicast protocols. DIVE provides a dynamic virtual environment where applications and users can enter and leave the environment on demand. Several user-related abstractions have been introduced to ease the task of application and user interface construction.

1 Background

The VR-related research at SICS is focused on distribution, collaboration, interaction and multi-user aspects of virtual reality. In order to perform research in these areas, we need a fully distributed software platform where several participants can meet and interact in a shared virtual environment. We also want the platform to be reasonably "open", so that new applications easily can be prototyped and tested. From these goals we have developed a software platform for virtual environment systems with specific focus on multi-user, distribution and human-computer interaction support. The platform, the Distributed Interactive Virtual Environment, DIVE, has enabled us to perform a multitude of experiments with virtual environments.

The DIVE system has been developed at the Distributed Systems Laboratory, Swedish Institute of Computer Science with the aid of the Interaction and Presentation Laboratory, Royal Institute of Technology. The work has been done within the MultiG program, a Swedish national research effort on high speed communication and distributed applications.

0-7803-1363-1/93 $3.00 © 1993 IEEE

2 The DIVE system

2.1 Introduction

A participant in a DIVE virtual world is either a human user or an application process. Users navigate in 3D space and may see, meet and collaborate with other users and applications in the environment. Users are represented by graphical objects called *body-icons*. Body-icons may be used as templates on which users' input devices are graphically modeled in 3D space. Also, the body-icon facilitates awareness of ongoing activities since it defines the position from which a user sees the world.

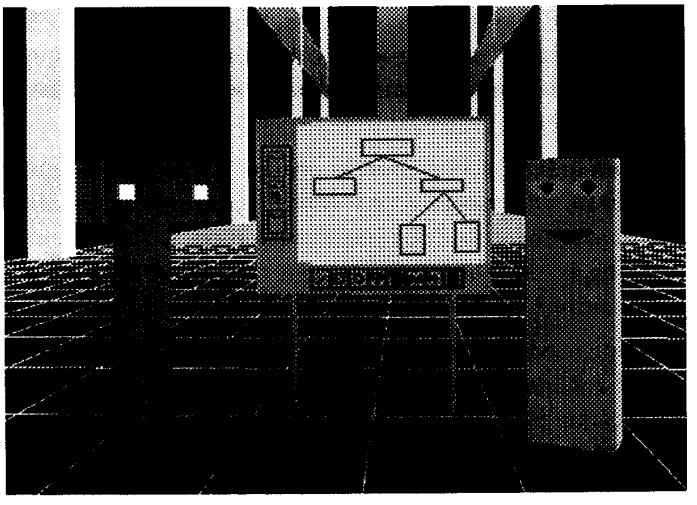

Figure 1: Two users and an application in a virtual environment.

As an example, Figure 1 illustrates two simple body-icons representing users in a synthetic world, as viewed from a third user. The users are standing in front of a "white-board" application.

A user sees and interacts with a world through an interface application called a *visualizer*. A visualizer can be set up to accommodate a wide range of I/O devices such as HMDs, wands, datagloves, etc. It reads the user's input devices and maps the physical actions taken by the user to actions in the DIVE system. This includes navigation in 3D space, selecting and grasping objects, etc.

In addition to user processes, any number of application processes may exist within a world. When started, an application process typically builds its user interface by creating and introducing graphical objects in the world. Thereafter, the process listens to world events and messages. When an event occurs or a message is received, it reacts according to some control logic.

2.2 Basic model: worlds and processes

Conceptually, the distribution model is best described as a memory shared over a network. A set of processes interact by making concurrent accesses to the shared memory and by sending messages to each other.

The memory, or database, is partitioned into *worlds*. Each world represents a specific set of objects and parameters completely distinct from other worlds. A DIVE process is a member of exactly one such world at a time, although it may change worlds dynamically. Since a world represents a local "state", a change of world results in a complete switch of context.

A DIVE *process* represents either a human user or an application. From a system point of view, there is no distinction between the two. A process acts on a world by modifying objects and parameters and by sending messages to the other processes of that world. Internally, each process runs multiple light weight processes, each thread being responsible for a certain task such as rendering, controlling I/O devices or updating the database.

Worlds are implemented by ISIS *process groups* [4]. A process group is a set of processes which may be addressed as one entity: messages addressed to the group are relayed by multicast protocols. The prime advantage with the process group abstraction in our setting is the ability to manage replicated data with atomic multicast protocols and achieve high availability by fault-tolerant computing algorithms.

A world is implemented as a process group where each participating process actively manages its own replica; a complete copy of the database kept in primary memory. From an application point of view the replication is transparent, a process only modifies an object in what it sees as a shared database.

The concurrency control mechanism uses mutual exclusive locks, reliable source ordered multicast and distributed object locks to ensure consistency between the copies of the replicated database. For a more detailed description, see [6].

2.3 Objects

A DIVE *object* is the elementary information bearer in the replicated database. Each object has a globally unique identifier which is used for global referencing. Some other fields of an object are the following:

- **Geometrical transformation:** Objects can be composed hierarchically. In addition to its own transformation, each object inherits the rotation and translation of the object on the level above.
- **Status flags:** The flags control basic properties such as how an object is displayed and interacted with. For example, there is a "grasp" flag that controls whether an object may be grasped by a user.

- **Views:** Each object may have a number of graphical representations, *views*. Some of the currently implemented views are: lines, spheres, cylinders, text-strings, boxes, grids, pixmaps and polygons. Since an object may have multiple views, a rendering process may change the object's graphical representation "on the fly". For example, an object may be represented by a simpler view if it is seen from a distance.

- **Behaviour:** A simple reactive behaviour can be associated with an object without having an application controlling it. This is done by associating a finite state machine with the object. Each transition in the state machine may be triggered by a message, resulting in changes in transformation, material, etc.

2.4 Persons and interaction

The user interface in DIVE uses the concept of *persons* to model a number of user interface abstractions. One such such abstraction is the *body-icon*, which is moved around in 3D space by the user. A *head* is a part of the body-icon which may be coupled directly to the movement of a HMD. One or two *eyes* identify viewpoints from which the visualizer renders the world. Many additional objects may be associated to persons, such as *ears*, *hands* and *visors*.

A *visor* is a transparent rectangular object placed just in front of an eye, where information and virtual control devices may be placed. Since the visor is placed at a short distance from an eye, it is for practical purposes invisible to other users.

Users can *select* and *grasp* objects by using interaction devices. When a user grasps an object, it is typically attached to the user's body-icon in some way, for example to the icon's hand. A select is normally a notification for the object or other processes to react to. In both cases, a message is distributed to all members of the world. This message can then be interpreted by an application, or alternatively, the object can react autonomously according to its behaviour description.

2.5 Vehicles

A user must be able to move through a virtual world and interact with applications, objects and other participants. In DIVE, this is accomplished with the aid of the various *vehicles*. A vehicle maps physical input devices to actions in the virtual environment, such as movement of a user's body-icon, the sending of messages and the grasping of objects.

There exists several vehicles in DIVE. One is the *mouse* vehicle, which supports movement in six degrees of freedom and basic 3D interaction through the X11 window and mouse interface. Movement is accomplished by dividing the different degrees of freedom into pairs and mapping them onto three icons. The icons

are presented in the center of the display window and the user moves by drawing a rubberband from the selected icon. The user can also select and grasp objects simply by placing the mouse pointer on the object and pressing a mouse button.

Another example is the *HMD* vehicle, which is designed to be used with a head mounted display and a "flying mouse", both equipped with Ascension Technology magnetic trackers. The sensor on the HMD tracks the movements of a user's head. The flying mouse is used for movement by pointing it in the desired direction and pressing a button. Other buttons are used to select and grasp objects by aiming a virtual beam at the object and then pressing the appropriate button.

2.6 Hardware and implementation

The DIVE system is written in C. Currently, DIVE runs on SUN4s with GX, GS or GT graphics hardware, IBM RS6000s with graphics hardware that supports GL and on Silicon Graphics workstations. The system is heterogeneous since users on different platforms can be present in the same virtual environment and work with applications running on any platform. The graphics may be displayed with various degrees of rendering quality, ranging from gouraud-shaded z-buffered polygons to simple wireframe rendering. Note that machines not equipped with graphics hardware can still run non-rendering applications.

3 Applications in DIVE

One of the fundamental principles in DIVE is to use the advantages a distributed platform can offer. One such advantage is the separation of the user's interfaces to the virtual environment and the semantics supplied by an application. In DIVE, a visualizer is an example of a virtual environment interface, while an application (a separate process) typically manifests itself by modifying the content of the shared world database, and reacts to events taking place in the world. Application specific information regarding objects in the database is maintained by the application itself. This information is not distributed to other processes, as the semantics of the information is local to the application.

Several applications have been built in DIVE. Some of those use traditional click-and-drag interfaces, such as Multidraw [10], where a 2-dimensional multi-user drawing application is projected within the 3D environment. Other applications use invisible, implicit interfaces, such as in the family of Aura [8] [3] applications, where the intersection of invisible geometrical volumes around users and objects trigger actions, such as an audio conference. Also, a direct manipulation robot interface [1], a 3D graph editor and an experimental 3D air traffic control system has been built using the system.

4 Related Work

The 3D navigation technique we employ in our screen-and-mouse interface is based on [9]. Other multi-user VR systems have been developed, e.g. Rubber Rocks [2], RB2 [5]. A difference between these and our approach is that DIVE can accommodate any number of users running several different applications without reconfiguration. The VEOS system [7] is somewhat similar to DIVE in that it uses peer to peer distribution between processes as opposed to Rubber Rocks which uses a client/server architecture. Our vehicle concept has similarities to the dialogue managers in Rubber Rocks.

5 Conclusions and Future Work

We have presented an experimental software environment for the development of multi-user virtual reality applications, in which a rich scale of experiments have been performed. We have less accentuated the graphic aspects of virtual environments and instead focused on multi-user and 3D interaction aspects.

The use of active replication has proven a powerful method and model for virtual worlds. In fact, once past the initial threshold of learning the process group abstraction, many application developers find this model appealing, and in some ways more intuitive than a traditional client/server model. The use of active replication has also resulted in a degree of fault-tolerance and persistency. A natural consequence of the distribution model is the ability to separate the virtual environment interface from the applications by using separate peer processes: visualizers and application processes, respectively. Application processes can then act on worlds completely independent of I/O device issues or the number of users, for example.

We have found that the effort of designing virtual worlds and applications is huge. It is therefore essential to develop tools and abstractions which may aid the design of virtual environments. We have introduced some "higher level" concepts, such as vehicles, visors and behaviours which we have found helpful when designing virtual reality applications. However, we see this only as a first effort and that more research is needed in this area.

A question which we have not addressed in full is what application areas are best served by an active replication based distributed VR system such as DIVE. With our model, we have found that many user interfaces which can be modeled by direct real-world metaphors are ideal. Simulation-like applications, terrain modeling etc., have been less successful because of the large amount of data in such applications. However, we are currently addressing this issue as well, where terrain or background object serve as "second class" objects which are of a static nature and cannot directly be interacted with.

Note

The DIVE software is freely available for non-commercial purposes. Contact the authors for more information.

References

[1] Magnus Andersson, Lennart E. Fahlén, and Torleif Söderlund. A virtual environment user interface for a robotic assistive device. In *Proceedings of the second European Conference on the Advancement of Rehabilitation Technology*, pages 33–57, Stockholm, May 1993.

[2] Perry A. Appino, Bryan J. Lewis, Lawrence Koved, Daniel T. Ling, David A. Rabenhorst, and Christopher F. Codella. An architecture for virtual worlds. *Presence*, 1(1), 1991.

[3] Steve Benford and Lennart E. Fahlén. A spatial model of interaction in large virtual environments. In *Proceedings of ECSCW '93*, September 1993.

[4] Kenneth P. Birman, Robert Cooper, and Barry Gleeson. Programming with process groups: Group and multicast semantics. Technical Report TR-91-1185, Dept. of Computer Sciences, University of Cornell, January 1991.

[5] Chuck Blanchard, Scott Burgess, Young Harville, Jaron Lanier, Ann Lasko, Mark Oberman, and Michael Teitel. Reality built for two: A virtual reality tool. *ACM SIGGRAPH Computer Graphics*, 24(2):35–36, 1992.

[6] Christer Carlsson and Olof Hagsand. DIVE – a platform for multi-user virtual environments. *Computers and Graphics*, 6, 1993.

[7] Geoffrey P. Coco. Veos 2.0 tool builders manual. Technical report, Human Interface Technology Lab, University of Washington, 1992.

[8] Lennart E. Fahlén, Charles G. Brown, Olov Ståhl, and Christer Carlsson. A space based model for user interaction in shared synthetic environments. In *INTERCHI '93 conference proceedings*, Amsterdam, April 1993.

[9] Jock D. Macinlay, Stuart K. Card, and George G. Robertson. Rapid controlled movement through a virtual 3D workspace. In *SIGGRAPH '90 conference proceedings*, pages 171–176, Dallas, August 1990.

[10] Olov Ståhl. Mdraw - a tool for cooperative work in the MultiG telepresence environment. Technical Report SICS-T92:05, SICS, 1992.